Protocole TCP

Table des matières

TCP est l'acronyme de « Transmission Control Protocol », il est défini dans la RFC 793. Les données encapsulées dans un en-tête TCP sont des paquets TCP.

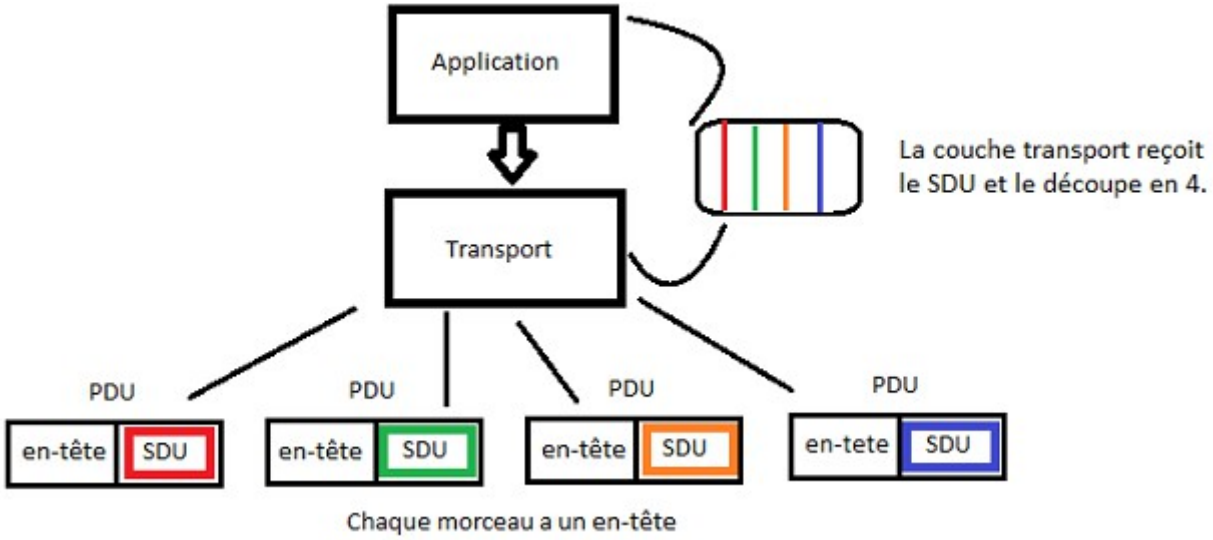
1. TCP – Transmission Control Protocol

1.1. Introduction

Le rôle de la couche transport est de rendre possible la communication logique entre applications. En anglais, on parle de logical end-to-end communication.

Il y a une grande différence entre la communication logique établie par la couche transport et celle établie par la couche réseau. La couche transport établit une communication logique entre les processus d'applications alors que la couche réseau établit une communication logique entre les hôtes.

Quand un hôte envoie un message, il utilise une application comme un client de messagerie. Cette application lui donne accès aux services réseaux pour envoyer un mail. Une fois ce SDU reçu par la couche transport, il sera converti en un PDU. La couche transport, qui est aussi responsable de la fragmentation des unités de données, va « couper » ce SDU en plusieurs chaînes (ou morceaux) qu'on appelle « chunks » en anglais. À chaque « morceau » sera ajouté un en-tête. Le SDU reçu par la couche transport sera donc « brisé » en 4 PDU distincts, par exemple :



Ces 4 PDU seront envoyés à la couche N-1 (la couche réseau en l'occurrence).

1.2. Caractéristiques de TCP

Cinq points principaux caractérisent ce protocole :

1. TCP contient un mécanisme pour assurer **le bon acheminement des données**. Cette possibilité est absolument indispensable dès lors que les applications doivent transmettre de gros volumes de données et de façon fiable.

   Il faut préciser que les paquets de données sont acquittés de bout en bout et non de point en point. D'une manière générale le réseau assure l'acheminement et les extrémités le contrôle.

2. Le protocole TCP permet l'établissement d'**un circuit virtuel** entre les deux points qui échangent de l'information. On dit aussi que TCP fonctionne en mode connecté (par opposition à UDP qui est en mode non connecté ou encore mode datagramme).

- Avant le transfert les 2 applications se mettent en relation avec leurs OS respectifs, les informent de leurs désirs d'établir ou de recevoir une communication.
- Pratiquement, l'une des deux applications doit effectuer un appel que l'autre doit accepter.
- Les protocoles des 2 OS communiquent alors en s'envoyant des messages au travers du réseau pour vérifier que le transfert est possible (autorisé) et que les deux applications sont prêtes pour leurs rôles.
- Une fois ces préliminaires établis, les modules de protocole informent les applications respectives que la connexion est établie et que le transfert peut débuter.
- Durant le transfert, le dialogue entre les protocoles continue, pour vérifier le bon acheminement des données.

Conceptuellement, pour établir une connexion -- un circuit virtuel -- il faut avoir réunis les éléments du quintuplet :

Le protocole
C'est TCP mais il y pourrait y avoir d'autres transports qui assurent le même service...

IP locale
Adresse de la machine qui émet.

Port local
Le numéro de port associé au processus. Il est imposé ou est déterminé automatiquement comme nous le verrons dans le cours de programmation.

IP distante
Adresse de la machine distante.

Port distant
Le numéro de port associé au service à atteindre. Il est obligatoire de le connaître précisément.

L'ensemble de ces cinq éléments définit un circuit virtuel unique. Que l'un d'eux change et il s'agit d'une autre connexion !

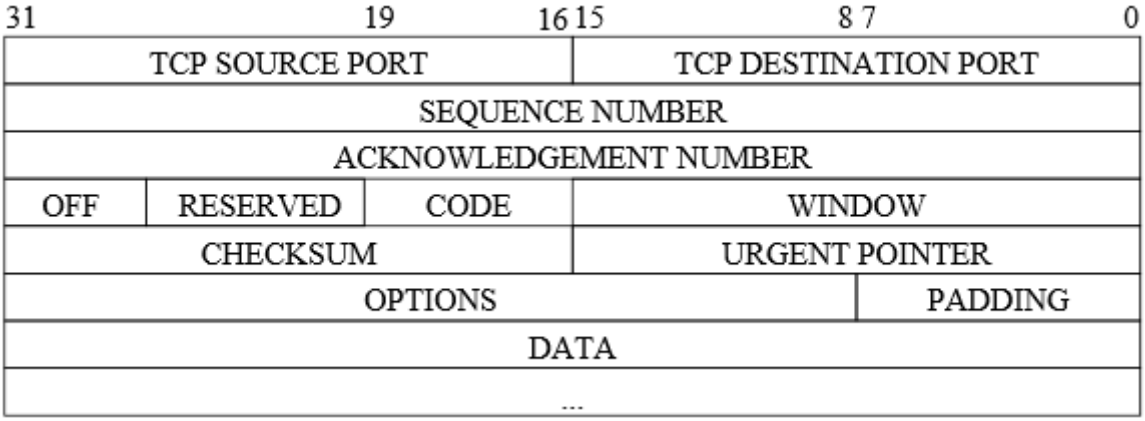
3. TCP a la capacité de **mémoriser des données** :
   - Aux deux extrémités du circuit virtuel, les applications s'envoient des volumes de données absolument quelconques, allant de 0 octet à des centaines (ou plus) de Mo.
   - À la réception, le protocole délivre les octets exactement comme ils ont été envoyés.
   - Le protocole est libre de fragmenter le flux de données en paquets de tailles adaptées aux réseaux traversés. Il lui incombe cependant d'effectuer le réassemblage et donc de stocker temporairement les fragments avant de les présenter dans le bon ordre à l'application.
4. TCP est **indépendant vis à vis des données transportées**, c'est un flux d'octets non structuré sur lequel il n'agit pas.
5. TCP simule une connexion en **full duplex**. Pour chacune des deux

applications en connexion par un circuit virtuel, l'opération qui consiste à lire des données peut s'effectuer indépendamment de celle qui consiste à en écrire.

Le protocole autorise la clôture du flot dans une direction tandis que l'autre continue à être active. Le circuit virtuel est rompu quand les deux parties ont clos le flux.

1.3. Description de l'en-tête

La figure suivante montre la structure d'un en-tête TCP. Sa taille normale est de 20 octets, à moins que des options soient présentes.



TCP SOURCE PORT

Le numéro de port de l'application locale.

TCP DESTINATION PORT

Le numéro de port de l'application distante.

SEQUENCE NUMBER

C'est un nombre qui identifie la position des données à transmettre par rapport au segment original. Au démarrage de chaque connexion, ce champ contient une valeur non nulle et non facilement prévisible, c'est la séquence initiale ou ISN.

TCP numérote chaque octet transmis en incrémentant ce nombre 32 bits non signé. Il repasse à 0 après avoir atteint $2^{32} - 1$ (4 294 967 295).

Pour le premier octet des données transmis ce nombre est incrémenté de un, et ainsi de suite...

ACKNOWLEDGEMENT NUMBER

C'est un numéro qui identifie la position du dernier octet reçu dans le flux entrant.

Il doit s'accompagner du drapeau ACK (voir plus loin).

OFF

pour OFFSET, il s'agit d'un déplacement qui permet d'atteindre les données quand il y a des options. Codé sur 4 bits, il s'agit du nombre de mots de 4 octets qui composent l'en-tête. Le déplacement maximum est donc de 60 octets ($2^4 - 1 \times 4$ octets). Dans le cas d'un en-tête sans option, ce champ porte la valeur 5. 10 mots de 4 octets sont donc possibles pour les options.

RESERVED

Six bits réservés pour un usage futur !

CODE

Six bits pour influer sur le comportement de TCP en caractérisant l'usage du segment :

URG	Le champ "URGENT POINTER" doit être exploité.
ACK	Le champ "ACNOWLEDGMENT NUMBER" doit être exploité.
PSH	C'est une notification de l'émetteur au récepteur, pour lui indiquer que toutes les données collectées doivent être transmises à l'application sans attendre les éventuelles données qui suivent.
RST	Re-initialisation de la connexion
SYN	Le champ "SEQUENCE NUMBER" contient la valeur de début de connexion.
FIN	L'émetteur du segment a fini d'émettre.

WINDOW

Le flux TCP est contrôle de part et d'autre pour les octets com- pris dans une zone bien délimitée et nommée " fenêtre ". La taille de celle-ci est définie par un entier non signé de 16 bits, qui en limite donc théoriquement la taille `a 65 535 octets.

Chaque partie annonce ainsi la taille de son buffer de réception. Par construction, l'émetteur n'envoie pas plus de données que le récepteur ne peut en accepter.

Cette valeur varie en fonction de la nature du réseau et surtout de la bande passante devinée à l'aide de statistiques sur la valeur du RTT.

CHECKSUM

Un calcul qui porte sur la totalité du segment, en-tête et données.

URGENT POINTER

Ce champ n'est valide que si le drapeau URG est armé. Ce pointeur contient alors un offset à ajouter à la valeur de SEQUENCE NUMBER du segment en cours pour délimiter la zone des données urgentes à transmettre à l'application.

Le mécanisme de transmission à l'application dépend du système d'exploitation.

OPTIONS

C'est un paramétrage de TCP. Sa présence est détectée dès lors que l' OFFSET est supérieur à 5.

Les options utilisées :

mss	La taille maximale du segment 5 des données applicatives que l'émetteur accepte de recevoir. Au moment de l'établissement d'une connexion (paquet comportant le flag SYN), chaque partie annonce sa taille de MSS. Ce n'est pas une négociation. Pour de l'Ethernet la valeur est 1460 (= MTU − 2 × 20).
timestamp	pour calculer la durée d'un aller et retour (RTT ou "round trip time").
Wscale	Facteur d'échelle ("shift") pour augmenter la taille de la fenêtre au delà des 16 bits du champ WINDOW (> 65535). Quand cette valeur n'est pas nulle, la taille de la fenêtre est de 65535×2^{shift} . Par exemple si "shift" vaut 1 la taille de la fenêtre est de 131072 octets soit encore 128 ko.
nop	Les options utilisent un nombre quelconque d'octets par

contre les paquet TCP sont toujours alignés sur une taille de mot de quatre octets ; à cet effet une option “No Operation” ou nop, codée sur 1 seul octet, est prévue pour compléter les mots.

PADDING

Remplissage pour se caler sur un mot de 32 bits.

DATAS

Les données transportées. Cette partie est de longueur nulle à l’établissement de la connexion, elle peut également être nulle par choix de l’application.

1.4. Les ports

Le numéro de port est le numéro qui permet de faire la distinction entre les applications. Par exemple, dans la transmission d'un mail, le premier service ou la première application utilisée est un MUA. Le MUA utilise le protocole SMTP pour envoyer le mail au serveur de messagerie souvent en passant par un MSA. Dans l'ordre de réception, on utilisait également un autre MUA pour retirer le mail du serveur de messagerie avec un protocole de réception comme IMAP ou POP. Il se peut donc que les deux services (POP/IMAP et SMTP) soient exécutés au même moment sur une même machine hôte. C'est là qu'intervient le numéro de port, qui permet de faire la distinction entre les services qui ont été demandés par l'application distante, qu'il s'agisse d'un serveur de messagerie ou d'un client de messagerie. Le protocole SMTP utilise le protocole TCP pour la transmission au numéro de port 25.

L'organisme IANA (Internet Assigned Numbers Authority) classe les numéros de port en trois catégories principales, comme l'illustre le tableau ci-dessous :

Portée	Catégorie	Description
0 – 1023	Ports bien connus	Ports réservés pour des services bien connus (web, envoi de mail, etc.).
1024 – 49151	Ports réservés	Ports réservés pour être utilisés par des applications propriétaires.
49152 – 65635	Ports dynamiques	Ports « libres » à utiliser pour vos applications. Ils ne sont ni pour des services bien connus, ni réservés par une entreprise quelconque.

Voici quelques ports bien connus :

Protocole	Description	Protocole de transmission	Numéro de port	Statut d'assignation
File Transfert Protocole (FTP)	Protocole de transfert de fichier	TCP	21	Officiel
Secured SHell (SSH)	Protocole permettant l'échange de données par le biais d'un canal sécurisé	TCP & UDP	22	Officiel
Telnet	Utilisé pour l'établissement des	TCP	23	Officiel

	sessions à distance			
Simple Mail Transfer Protocol (SMTP)	Protocole d'envoi de courrier électronique	TCP	25	Officiel
WHOIS protocol	Protocole ou service utilisé pour l'identification d'une machine par son nom de domaine ou son adresse IP. La procédure d'identification se fait par une requête envoyée à un des registres Internet pour obtenir des informations.	TCP	43	Officiel
Domain Name System (DNS)	Protocole de résolution des noms de domaine	TCP & UDP	53	Officiel
HyperText Transfer Protocol (HTTP)	Protocole de téléchargement (principalement de pages web)	TCP & UDP	80	Officiel
Post Office Protocol Version 2 (POP2)	Protocole de retrait de mails d'un serveur de messagerie	TCP	109	Officiel
Post Office Protocol Version 3 (POP3)	Protocole de retrait de mails d'un serveur de messagerie	TCP	110	Officiel
Internet Message Access Protocol (IMAP)	Protocole de retrait et consultation de mails d'un serveur de messagerie	TCP & UDP	143	Officiel
…	…	…	…	...

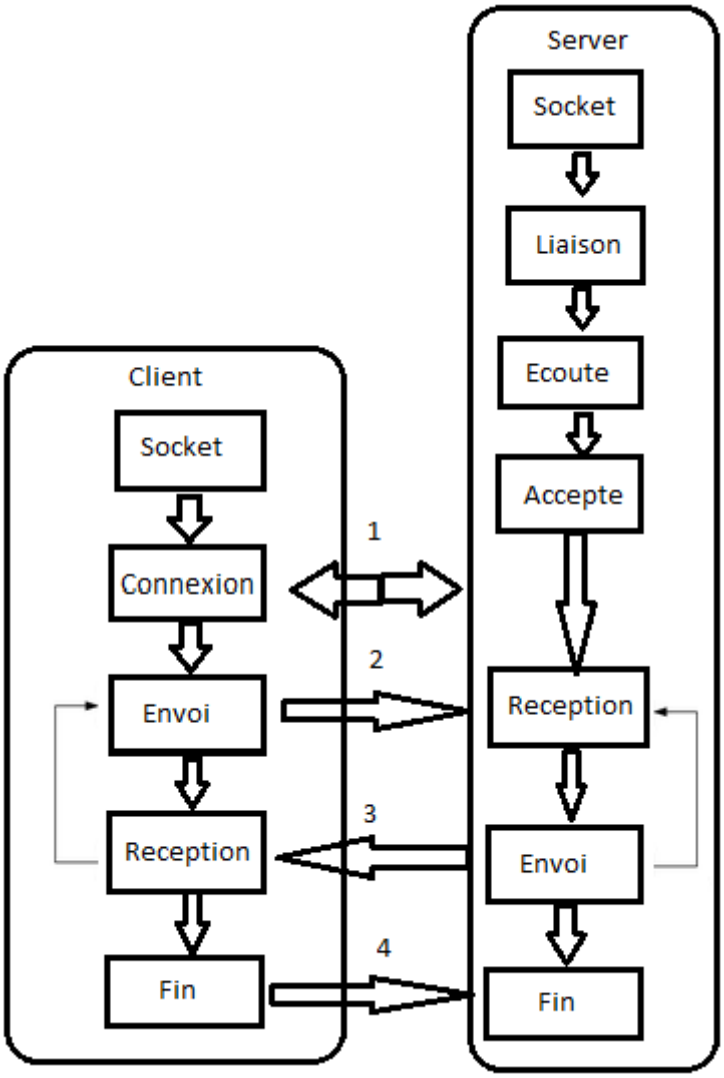
- Statut d'assignation : le statut officiel veut dire que le couple application / numéro de port a été enregistré dans les registres de l'IANA. En d'autres termes, il est défini par une convention que telle application utilise tel numéro de port.
- Protocole de transmission : les protocoles peuvent s'utiliser entre eux. Le protocole SMTP qui sert à envoyer un mail s'appuie sur le protocole TCP pour le transmettre. On parle de sous-couchage de protocoles (underlayering protocols).

1.5. Les sockets

Les protocoles se chevauchent : un protocole applicatif (SMTP, POP, HTTP, etc) peut être interfacé à un protocole de transport (UDP, TCP). Un socket est une interface entre les processus : en réseau, un socket sert donc à faire communiquer un processus avec un service qui gère le réseau. Chaque socket a une

adresse de socket. Cette adresse est constituée d'une adresse IP et d'un numéro de port. C'est grâce à la programmation de socket que l'on définit le modèle de communication. Si le socket a été configuré de manière à envoyer ou recevoir, c'est un modèle Half-Duplex. S'il a été configuré de manière à envoyer et recevoir simultanément, il s'agit d'un modèle Full-Duplex. Étant donné que les sockets sont en fait une interface de programmation d'applications (API), on peut donc s'en servir pour programmer des applications en réseaux (par exemple, créer une application pour faire communiquer un client et un serveur).

Voici un schéma illustrant une communication entre un client et un serveur :



Le client commence à se connecter au serveur grâce aux sockets. Une fois la connexion établie (étape 1), le client et le serveur peuvent communiquer (étapes 2 et 3). À la fin de la communication, le client envoie une demande de terminaison de session au serveur (étape 4) et le serveur met fin à la connexion.

Le serveur utilise les sockets pour lier un port d'application à son processus correspondant. Ensuite, il « écoute » ce port. Ce faisant, il va découvrir qu'un client essaie de se connecter à lui par le numéro de port qu'il écoute. Il accepte donc la requête, établit la connexion (étape 1) et, finalement, les deux communiquent (étapes 2, 3 et 4). Le modèle de communication est Half-Duplex parce que le client envoie et attend la réponse du serveur et vice-versa.

1.6. Multiplexing / demultiplexing

Le rôle de la couche transport est d'acheminer ou de donner les PDU reçus aux processus d'application identifiables par un numéro de port. Il se peut que plusieurs services (processus d'applications) soient exécutés au même moment. Certaines applications peuvent avoir plusieurs instances en cours d'exécution qui donnent accès aux services du protocole HTTP. Lorsque la couche transport reçoit les PDU de la couche réseau, elle va examiner les en-têtes de ces PDU afin de retrouver l'identifiant du processus auquel le PDU doit être acheminé. C'est le démultiplexage : le fait de transmettre un PDU donné au processus d'une application donnée.

Ainsi, un protocole de la couche transport est responsable de la collection des SDU, de leur encapsulation, en spécifiant le numéro de port du processus de l'application utilisée et le numéro de port à utiliser pour le processus de l'application réceptrice. Cette encapsulation, c'est la transformation du SDU en PDU. Ce protocole est aussi responsable de la livraison de ce PDU à un protocole de la couche inférieure (couche réseau, protocole IP).

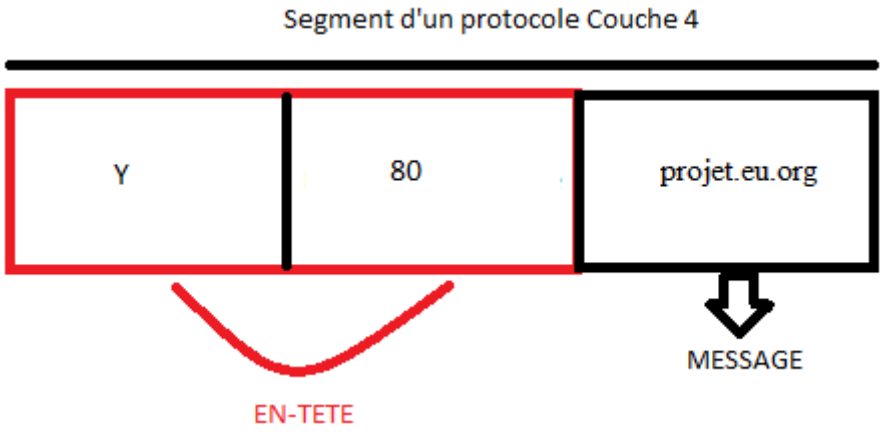
Les protocoles TCP et UDP sont donc responsables de la modification des en-têtes des unités de données lors du multiplexage / démultiplexage.

Un segment de protocole de transport est partiellement constitué des champs « port source » (source port), « port de destination » (destination port) et « SDU ». Le champ « port source » contiendra le numéro de port utilisé par l'application émettrice. Le champ « port de destination » contiendra le numéro de port identifiant l'application réceptrice. Finalement, le champ « SDU », c'est le message original.

Les serveurs web créent un nouveau processus pour chaque requête HTTP qu'ils reçoivent. Un serveur qui gère 10 requêtes gère donc 10 processus utilisant tous le même numéro de port.

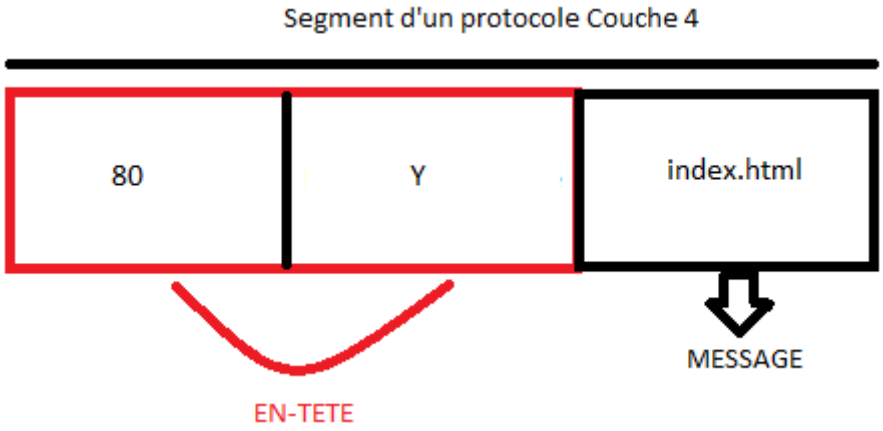
Pour le démultiplexage, il faut deux numéros de port (source et destination). En général, le protocole de cette couche (UDP ou TCP) va générer automatiquement un numéro de port qu'aucun processus n'utilise actuellement.

Y sera la valeur du champ « Source port » et 80 la valeur du champ « Destination port ».

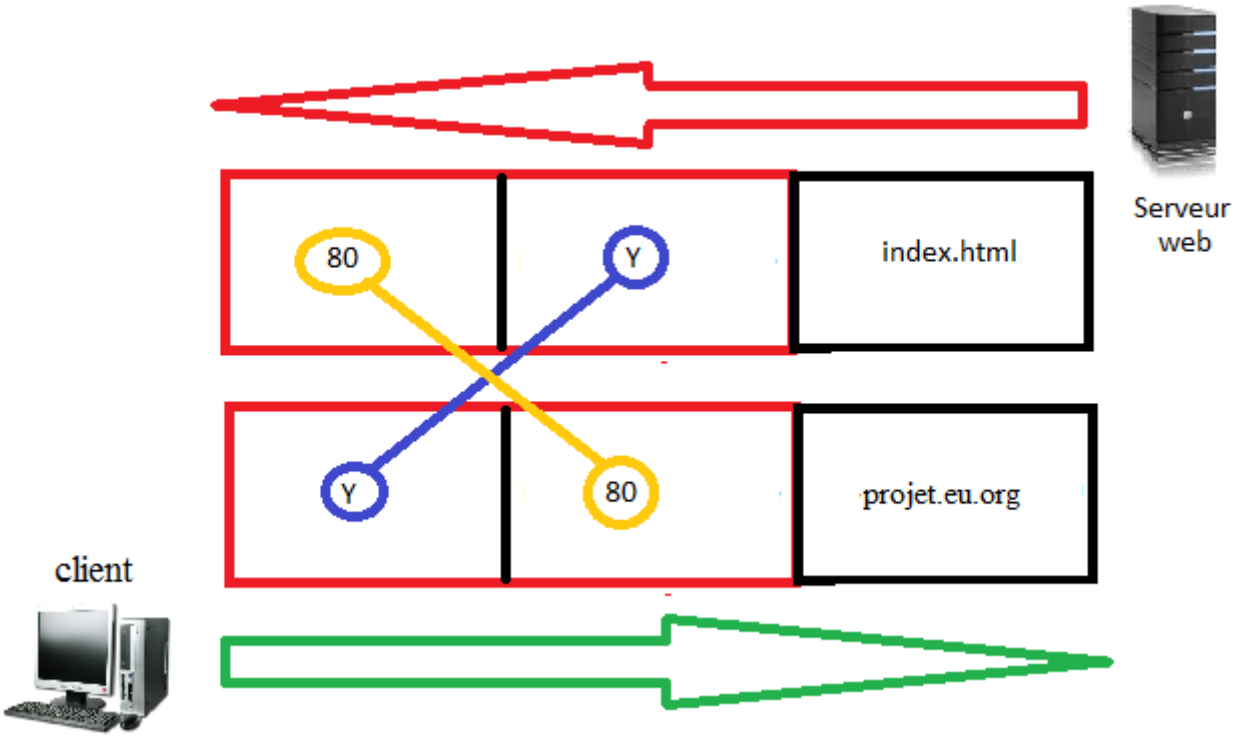


Le serveur va recevoir ce PDU et examiner sa constitution.

Dans la procédure d'envoi de la requête, le champ « Source port » avait pour valeur Y. Dans l'envoi de la réponse, le champ « Source port » prend la valeur du champ « Destination port » de la requête, et le champ « Destination port » prend la valeur du champ « Source port » de la requête. Ainsi, le segment envoyé par le serveur web ressemblera à ceci :



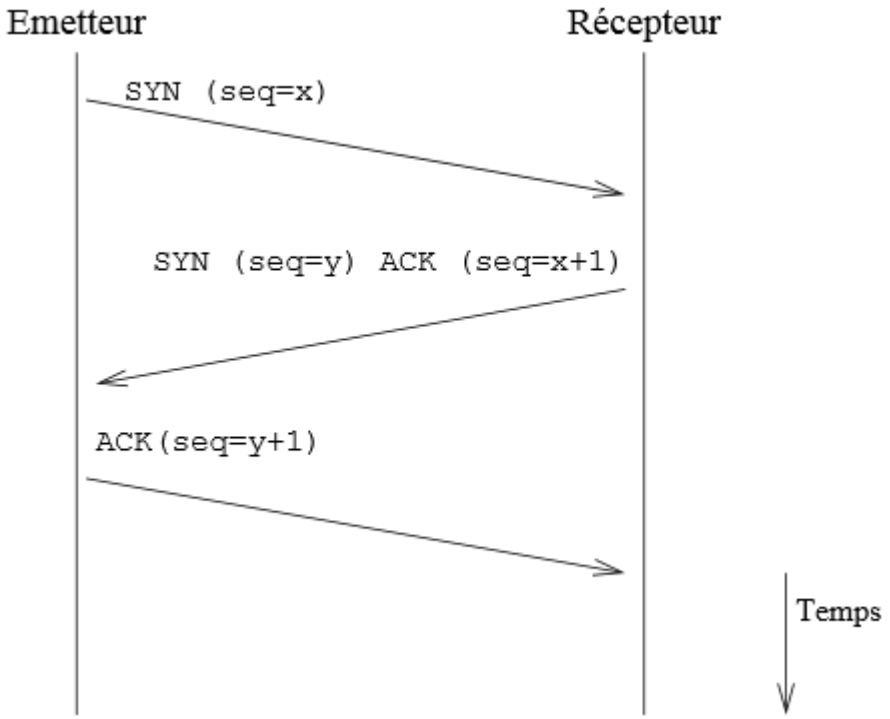
Pour résumer cela, voici un schéma qui montre les valeurs de chaque champ au niveau du client et au niveau du serveur :



2. Début et clôture d'une connexion

2.1. Établissement d'une connexion

L' établissement d'une connexion TCP s'effectue en trois temps :



On suppose que l' émetteur du premier paquet avec le bit SYN a connaissance du couple (adresse IP du récepteur, numéro de port du service souhaité).

L'émetteur du premier paquet est à l'origine de l'établissement du circuit virtuel, c'est une attitude généralement qualifiée de "cliente". On dit aussi que le client effectue une "ouverture active" (active open).

Le récepteur du premier paquet accepte l'établissement de la connexion, ce qui suppose qu'il était prêt à le faire avant que la partie cliente en prenne l'initiative. C'est une attitude de "serveur". On dit aussi que le serveur effectue une "ouverture passive" (passive open).

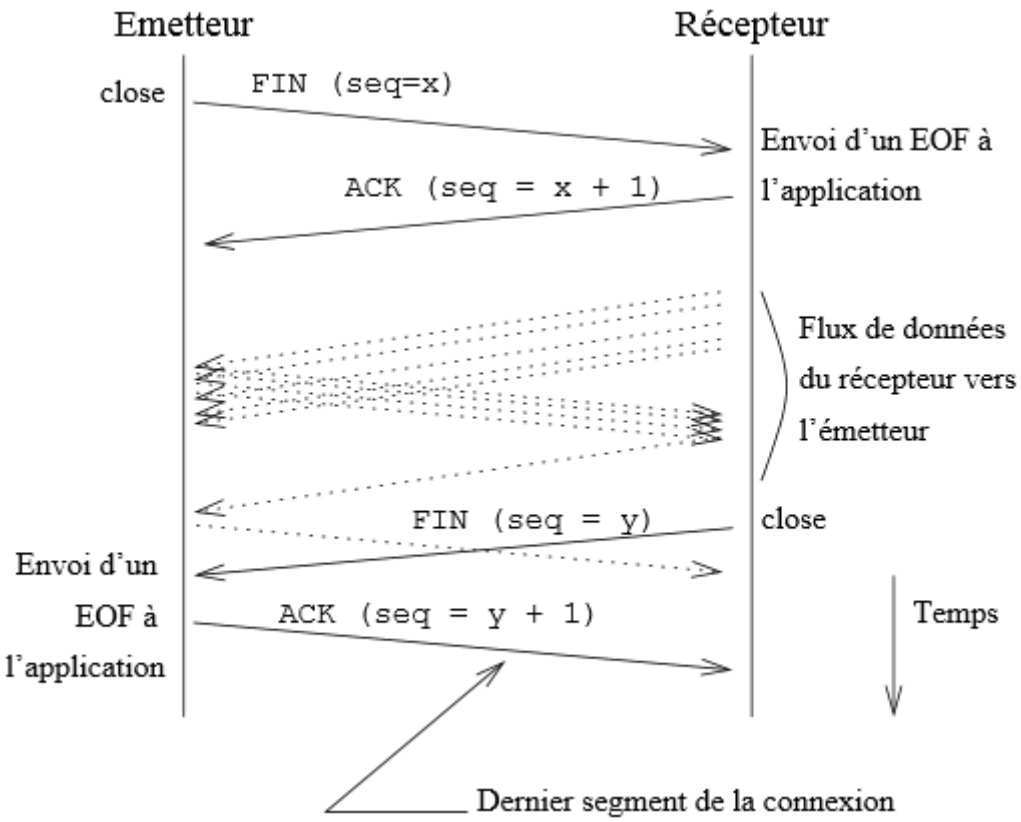
1. Le client envoie un segment comportant le drapeau SYN , avec sa séquence initiale (ISN = x).
2. Le serveur répond avec sa propre séquence (ISN = y), mais il doit également acquitter le paquet précédent, ce qu'il fait avec ACK (seq = x + 1).
3. Le client doit acquitter le deuxième segment avec ACK (seq = y + 1).

Une fois achevé cette phase nomme "three-way handshake", les deux applications sont en mesure d'échanger les octets qui justifient l'établissement de la connexion.

2.2. Clôture d'une connexion

2.2.1. Clôture canonique

Un change de trois segments est nécessaire pour l'établissement de la connexion ; il en faut quatre pour qu'elle s'achève de manière canonique ("orderly release").



La raison est qu'une connexion TCP est "full-duplex", ce qui implique que les données circulent indépendamment dans un sens et dans l'autre. Les deux directions doivent donc pouvoir être interrompues indépendamment l'une de l'autre.

L'application qui envoie un paquet avec le drapeau FIN indique à la couche TCP de la machine distante qu'elle n'enverra plus de donnée. La machine distante doit acquitter ce segment, comme il est indiqué sur la figure ci-dessus, en incrémentant d'une unité le "sequence number".

La connexion est véritablement terminée quand les deux applications ont effectué ce travail. Il y a donc échange de 4 paquets pour terminer la connexion.

Au total, sans compter les échanges propres au transfert des données, les deux couches TCP doivent gérer 7 paquets, il faut en tenir compte lors de la conception des applications !

Sur la figure on constate que le serveur continue d'envoyer des données bien que le client ait terminé ses envois. Le serveur a détecté cette attitude par la réception d'un caractère de EOF (en C sous Unix).

Cette possibilité a son utilité, notamment dans le cas des traitements distants qui doivent s'accomplir une fois toutes les données transmises, comme par exemple pour un tri.

2.2.2. Clôture abrupte

Au lieu d'un échange de quatre paquets comme précédemment, un mécanisme de reset est prévu pour terminer une connexion au plus vite (abortive release).

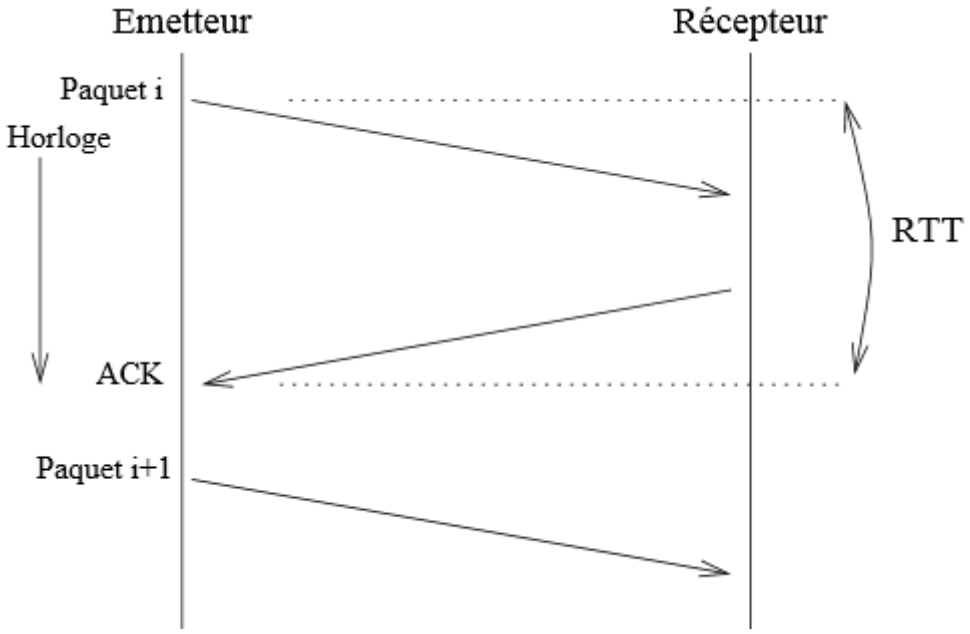
Ce type d'arrêt est typiquement géré par la couche TCP elle-même quand l'application est brutalement interrompue sans avoir effectué un appel à la primitive close, comme par exemple lors d'un appel à la primitive abort, ou après avoir rencontré une exception non prise en compte ("core dump"...).

L'extrémité qui arrête brutalement la connexion émet un paquet assorti du bit RST, après avoir (ou non) envoyé les derniers octets en attente. Ce paquet clôt l'échange. Il ne reçoit aucun acquittement. L'extrémité qui reçoit le paquet de reset (bit RST), transmet les éventuelles dernières données à l'application et provoque une sortie d'erreur du type "Connection reset per peer" pour la primitive de lecture réseau. Comme c'est le dernier échange, si des données restaient à transmettre à l'application qui a envoyé le RST elles peuvent être détruites.

3. Contrôle du transport

Le bon acheminement des données applicatives est assurée par un mécanisme d'acquittement des paquets.

3.1. Mécanisme de l'acquittement



- Au départ du Paquet i une horloge se déclenche. Si cette horloge dépasse une valeur limite avant réception de l'ACK le Paquet i est retransmis.

  Cette valeur limite est basée sur la constante MSL (Maximum Segment Lifetime) qui est un choix d'implémentation, généralement de 30 secondes à 2 minutes. Le temps maximum d'attente est donc de 2 × MSL.

- Le temps qui s'écoule entre l'émission d'un paquet et la réception de son acquittement est le RTT[1], il doit donc être inférieur à 2 × MSL. Il est courant sur l'Internet actuel d'avoir un RTT de l'ordre de la seconde. Il faut noter que le RTT est la somme des temps de transit entre chaque routeur et du temps passé dans les diverses files d'attente sur les routeurs.

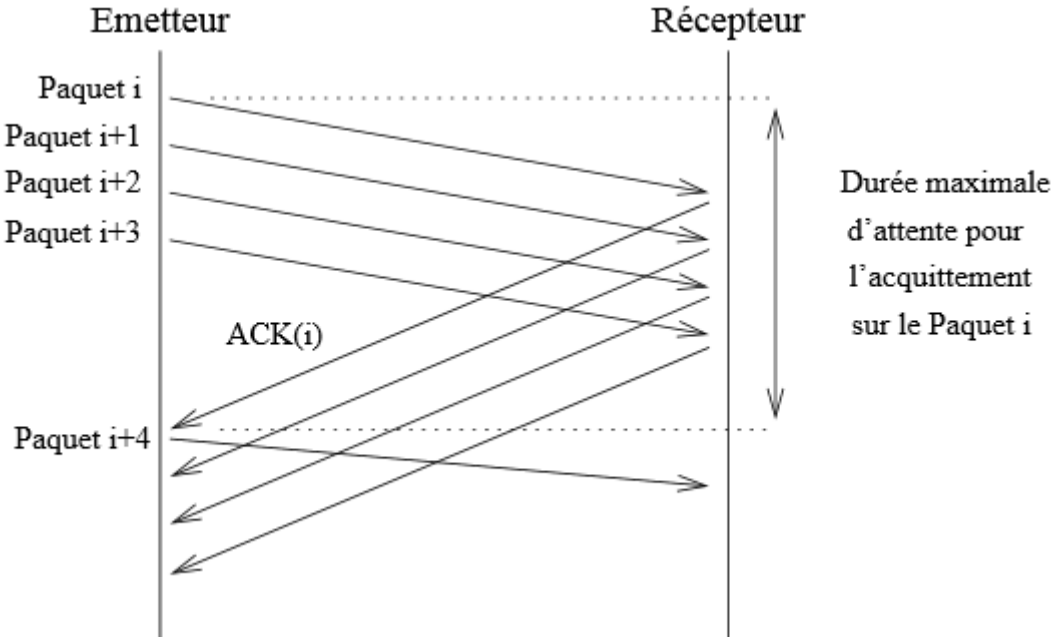
- L'émetteur conserve la trace du Paquet i pour éventuellement le renvoyer.

Si on considère des délais de transmission de l'ordre de 500 ms (voire plus), un tel mécanisme est totalement inadapté au transfert de flux de données. On peut aussi remarquer qu'il sous-emploie la bande passante du réseau.

1 Round Trip Time, calculé à l'aide de l'option timestamp

3.2. Fenêtres glissantes

Cette attente de l'acquittement est pénalisante, sauf si on utilise un mécanisme de fenêtres glissantes :



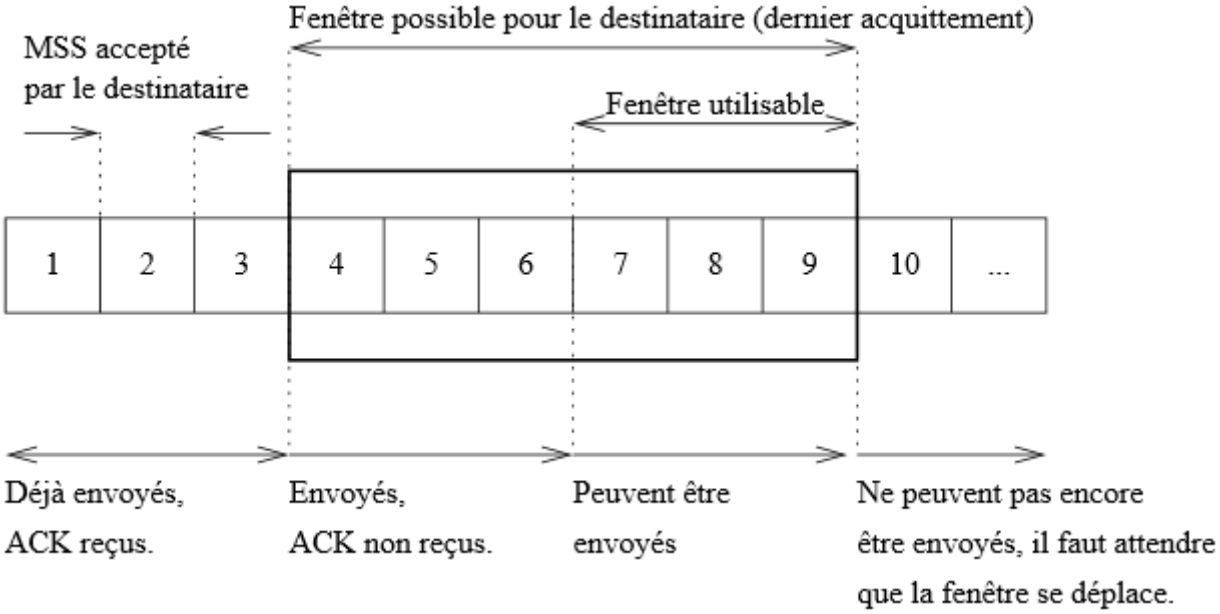
- Avec ce principe, la bande passante du réseau est beaucoup mieux employée.
- Si l'un des paquets doit être reémis, la couche TCP du destinataire aura toute l'information pour le replacer dans le bon ordre.
- À chaque paquet est associée une horloge comme sur la figure.
- Le nombre de paquets à envoyer avant d'attendre le premier acquittement est fonction de deux paramètres :

1. La largeur de la fenêtre change dynamiquement pour deux raisons :
   a) L'application change la taille de cette fenêtre.
   b) Chaque acquittement ACK envoyé est assorti d'une nouvelle valeur de taille de la fenêtre, permettant ainsi à l'émetteur d'ajuster à tout instant le nombre de segment qu'il peut envoyer simultanément. Celle valeur peut être nulle, comme par exemple lorsque l'application cesse de lire les données reçues. C'est ce mécanisme qui assure le contrôle de flux de TCP.
2. La taille maximale des données, ou MSS (Maximum Segment Size) vaut 512 octets par défaut. C'est la plus grande taille du segment de données que TCP enverra au cours de la session. Le datagramme IP a donc une taille égale au MSS augmentée de 40 octets (20 + 20), en l'absence d'option de TCP .

   Cette option apparaît uniquement dans un paquet assorti du drapeau SYN , donc à l'établissement de la connexion. Comme de bien entendu cette valeur est fortement dépendante du support physique et plus particulièrement du MTU (Maximum Transfer Unit).

   Sur de l'Ethernet la valeur maximale est $1500 - 2 \times 20 = 1460$, avec des trames l'encapsulation 802.3 de l'IEEE un calcul similaire conduit à une longueur de 1452 octets.

Chaque couche TCP envoie sa valeur de MSS en même temps que le paquet de synchronisation, comme une option de l'en-tête. Cette valeur est calculée pour éviter absolument la fragmentation de IP au départ des datagrammes.



Le débit obtenu dépend de la taille de la fenêtre et bien sûr de la bande passante disponible. Par contre l'agrandissement de la taille de la fenêtre ne se conçoit que jusqu'à une limite optimale au delà de laquelle des paquets sont perdus parce qu'envoyés trop rapidement pour être reçus par le destinataire. Or, pour fonctionner de manière optimale, TCP se doit de limiter au maximum la perte de paquets et donc leur réémission.

Cette taille limite optimale de la largeur de la fenêtre est, comme on peut le deviner, fonction de la bande passante théorique du réseau et surtout de son taux d'occupation instantané. Cette dernière donnée est fluctuante, aussi TCP doit-il asservir continuellement les tailles de fenêtre pour en tenir compte.

4. Exemple de paquets capturés

Le premier exemple montre un échange de paquets de synchronisation (SYN) et de fin (FIN) entre la machine clnt.chezmoi et la machine srv.chezmoi. L'établissement de la connexion se fait à l'aide de la commande telnet sur le port discard du serveur srv.chezmoi. La machine qui est à l'origine de l'établissement de la connexion est dite cliente, et celle qui est supposée prête à répondre, serveur. Pour information, le service discard peut être considéré comme l'équivalent du fichier /dev/null sur le réseau : les octets qu'on lui envoie sont oubliés ("discard").

L'utilisateur tape :

```
$ telnet srv discard
Trying...
Connected to srv.chezmoi.
Escape character is '^]'.
telnet> quit
Connection closed.
```

Et l'outil d'analyse réseau permet la capture pour l'observation des échanges

suivants. Le numéro qui figure en tête de chaque ligne a été ajouté manuellement, le nom de domaine “chezmoi” a été retiré, le tout pour faciliter la lecture :

```
0  13:52:30.274009 clnt.1159 > srv.discard: S 55104001:55104001(0) win 8192 <mss 1460>
1  13:52:30.275114 srv.discard > clnt.1159: S 2072448001:2072448001(0) ack 55104002 win 4096 <mss 1024>
2  13:52:30.275903 clnt.1159 > srv.discard: . ack 1 win 8192
3  13:52:33.456899 clnt.1159 > srv.discard: F 1:1(0) ack 1 win 8192
4  13:52:33.457559 srv.discard > clnt.1159: . ack 2 win 4096
5  13:52:33.458887 srv.discard > clnt.1159: F 1:1(0) ack 2 win 4096
6  13:52:33.459598 clnt.1159 > srv.discard: . ack 2 win 8192
```

Plusieurs remarques s’imposent :

1. Pour améliorer la lisibilité les numéros de séquences “vrais” ne sont indiqués qu’au premier échange. Les suivants sont relatifs. Ainsi le ack 1 de la ligne 2 doit être lu 2072448002 (2072448001 + 1). A chaque échange la valeur entre parenthèses indique le nombre d’octets échangés.
2. Les tailles de fenêtre (win) et de segment maximum (mss) ne sont pas identiques. Le telnet du client fonctionne sur HP-UX alors que le serveur telnetd fonctionne sur une machine BSD.
3. La symbole > qui marque le sens du transfert.
4. Le port source 1159 et le port destination discard.
5. Les flags F et S. L’absence de flag, repéré par un point.

Le deuxième exemple montre une situation de transfert de fichier avec l’outil ftp (File Transfer Protocol).

Il faut remarquer que l’établissement de la connexion TCP est ici à l’initiative du serveur, ce qui peut laisser le lecteur perplexe... L’explication est simple. En fait le protocole ftp fonctionne avec deux connexions TCP, la première, non montrée ici, est établie du client vers le serveur, supposé à l’écoute sur le port 21. Elle sert au client pour assurer le contrôle du transfert.

Lorsqu’un transfert de fichier est demandé via cette première connexion, le serveur établit une connexion temporaire vers le client. C’est cette connexion que nous examinons ici. Elle est clôturée dès que le dernier octet demandé est transféré.

Extrait du fichier /etc/services, concernant ftp :

```
ftp-data        20/tcp    #File Transfer [Default Data]
ftp-data        20/udp    #File Transfer [Default Data]
ftp             21/tcp    #File Transfer [Control]
ftp             21/udp    #File Transfer [Control]
```

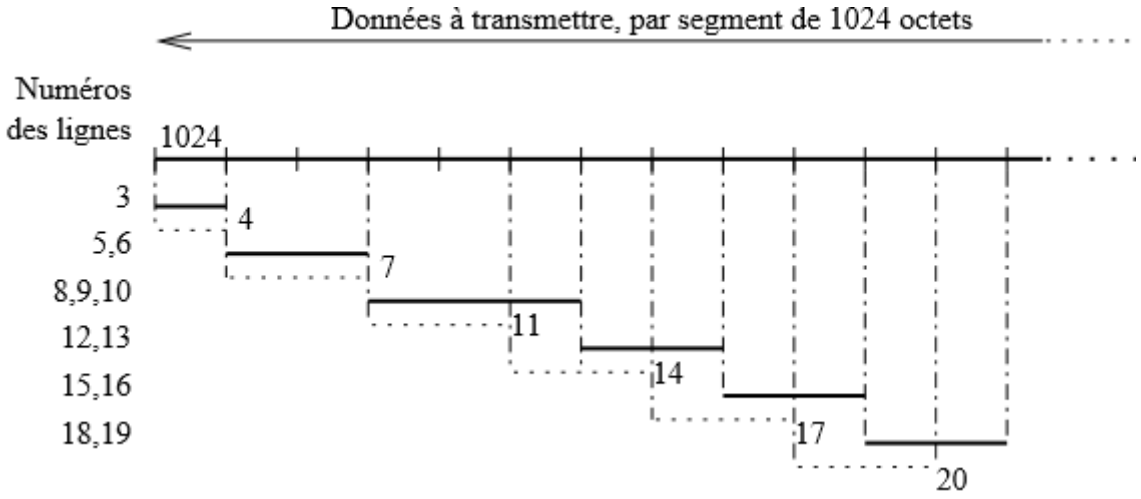
Dans cette exemple nous pouvons suivre le fonctionnement du mécanisme des fenêtres glissantes. Les lignes ont été numérotées manuellement et la date associée à chaque paquet supprimée.

```
0   srv.20 > clnt.1158: S 1469312001:1469312001(0) win 4096 <mss 1024> [tos 0x8]
1   clnt.1158 > srv.20: S 53888001:53888001(0) ack 1469312002 win 8192 <mss 1460>
```

```
2   srv.20 > clnt.1158: . ack 1 win 4096 [tos 0x8]
3   srv.20 > clnt.1158: P 1:1025(1024) ack 1 win 4096 [tos 0x8]
4   clnt.1158 > srv.20: . ack 1025 win 8192
5   srv.20 > clnt.1158: . 1025:2049(1024) ack 1 win 4096 [tos 0x8]
6   srv.20 > clnt.1158: . 2049:3073(1024) ack 1 win 4096 [tos 0x8]
7   clnt.1158 > srv.20: . ack 3073 win 8192
8   srv.20 > clnt.1158: . 3073:4097(1024) ack 1 win 4096 [tos 0x8]
9   srv.20 > clnt.1158: P 4097:5121(1024) ack 1 win 4096 [tos 0x8]
10  srv.20 > clnt.1158: P 5121:6145(1024) ack 1 win 4096 [tos 0x8]
11 clnt.1158 > srv.20: . ack 5121 win 8192
12  srv.20 > clnt.1158: P 6145:7169(1024) ack 1 win 4096 [tos 0x8]
13  srv.20 > clnt.1158: P 7169:8193(1024) ack 1 win 4096 [tos 0x8]
14  clnt.1158 > srv.20: . ack 7169 win 8192
15  srv.20 > clnt.1158: P 8193:9217(1024) ack 1 win 4096 [tos 0x8]
16  srv.20 > clnt.1158: P 9217:10241(1024) ack 1 win 4096 [tos 0x8]
17  clnt.1158 > srv.20: . ack 9217 win 8192
18  srv.20 > clnt.1158: P 10241:11265(1024) ack 1 win 4096 [tos 0x8]
19  srv.20 > clnt.1158: P 11265:12289(1024) ack 1 win 4096 [tos 0x8]
20  clnt.1158 > srv.20: . ack 11265 win 8192
... ...  ...
21  srv.20 > clnt.1158: P 1178625:1179649(1024) ack 1 win 4096 [tos 0x8]
22  clnt.1158 > srv.20: . ack 1178625 win 8192
23  srv.20 > clnt.1158: P 1212417:1213441(1024) ack 1 win 4096 [tos 0x8]
24  srv.20 > clnt.1158: P 1213441:1214465(1024) ack 1 win 4096 [tos 0x8]
25  srv.20 > clnt.1158: P 1214465:1215489(1024) ack 1 win 4096 [tos 0x8]
26  clnt.1158 > srv.20: . ack 1213441 win 8192
27  clnt.1158 > srv.20: . ack 1215489 win 8192
28  srv.20 > clnt.1158: P 1215489:1215738(249) ack 1 win 4096 [tos 0x8]
29  srv.20 > clnt.1158: F 1215738:1215738(0) ack 1 win 4096 [tos 0x8]
30  clnt.1158 > srv.20: . ack 1215739 win 8192
31  clnt.1158 > srv.20: F 1:1(0) ack 1215739 win 8192
32  srv.20 > clnt.1158: . ack 2 win 4096 [tos 0x8]
```

Remarques :

1. Le P symbolise le drapeau PSH. La couche TCP qui reçoit un tel paquet est informée qu'elle doit transmettre à l'application toutes les données reçues, y compris celles transmises dans ce paquet. Le positionnement de ce drapeau est à l'initiative de la couche TCP émettrice et non à l'application.
2. Le type de service ("Type Of service" tos 0x8) est demandé par l'application pour maximiser le débit.



5. Conclusion

Le protocole TCP a été conçu à une époque où l'usage de la commande ligne était universel, et les applications graphiques utilisant le réseau très rares !

Une trentaine d'années plus tard, on peut faire le constat pratiquement inverse : les applications textes interactives (beaucoup de petits messages applicatifs) disparaissent au profit d'applications moins interactives et qui sont plus orientées flux de données (vidéo, audio, téléphonie...) avec des échanges plus volumineux et des besoins en transport qui ont évolué.

Le principe de la fenêtre glissante, si performant qu'il soit pour assurer le bon acheminement des données, est bloquant pour certaines applications comme le web. En effet, si le paquet de données de tête n'est pas acquitté, les suivants, même reçus, sont en attente avant d'être délivrés à l'application.

Si la réponse comporte par exemple de nombreuses zones graphiques et textuelles différentes la fluidité de la consultation est considérablement amoindrie, et tenter de la compenser en établissant un grand nombre de connexions simultanées pour récupérer individuellement les éléments de la page, consomme beaucoup de ressources système et réseaux (celles de l'établissement des connexions) qui ne compense que partiellement ce soucis.

L'indépendance de TCP vis `a vis de la structure des données est également un inconvénient dans certaines applications comme la téléphonie pour la- quelle la notion de messages successifs est bien plus intéressante.

Depuis le début des années 2000 l'IETF met au point le protocole SCTP qui fournit des services similaires à ceux de TCP, en abandonne certains et apporte les nouvelles fonctionnalités adaptées aux nouveaux besoins.